

Grid-Enabled Desktop Environments: The GRENADE Project

Martyn Foster*, Daniel Hanlon†, Jon MacLaren†, James Marsh‡, Stephen Pettifer‡, **Stephen Pickles†**

†*Manchester Computing, University of Manchester, Oxford Road, Manchester. M13 9PL*

**Silicon Graphics Limited, 1530 Arlington Business Park, Theale, Reading, Berkshire. RG7 4SB*

‡*Department of Computer Science, University of Manchester, Oxford Road, Manchester. M13 9PL*

Abstract

Existing Grid middleware solutions have largely neglected to provide the type of desktop-integrated graphical user interfaces chosen by the vast majority of computer users. GRENADE tackles the issues associated with projecting the emerging fundamental activities of using the Grid onto the immensely successful graphical desktop paradigm. The first release of the GRENADE software demonstrates how standard grid functionality can easily and simply be accessed from the popular KDE desktop and partly from the Windows desktop. This approach yields immediate benefits as the novice user can almost unknowingly use the grid by harnessing familiar user interface paradigms which have been extended to take advantage of the Grid. Such extensions can also provide capabilities beyond what is achievable with the simple command line tools alone. Customisation for specific applications adds another dimension, further simplifying the remote execution of complex codes yet hiding the details that make up the grid.

1. Introduction

As the underlying technologies for forming computational Grids evolve and mature, increasingly sophisticated applications are being developed to harness the power of these distributed computing environments [1,2,3]. The diversity of these projects demonstrates the ability of the current infrastructure to provide secure remote access to data, processing power and specialist instrumentation. In spite of these successes, however, the vision of the Grid as enabling *coordinated resource and problem solving in dynamic, multi-institutional virtual organizations* [4] is still some way from being realised: far from being a flexible and dynamic relationship between the providers and consumers of generalised Grid resources, in the current situation we see many isolated Grid projects simply using the technology to provide connectivity between resources, much in the style of other distributed computing initiatives such as CORBA [5] or DCE [6]. The potential benefits of combining global scale authentication with brokering and sharing technology will only really be seen when the Grid is mature enough to allow these dynamic relationships to be formed and dissolved in a lightweight, cost effective manner.

Building and maintaining a Grid currently involves a significant investment of time and effort, and this presents a barrier to the widespread uptake of the technology. Much of this cost, however, can be attributed to the relative novelty of the software and the time required to understand the concepts and work around technological idiosyncrasies. These are not fundamental show-stoppers, and familiarity

and sound software engineering within the current Grid community will reduce these problems in time. A more important brake on the use of Grid technology by a wider audience is the means by which users interact with such systems. It is clear from the history of computing that widespread uptake of technology takes place at the point where it becomes easy to use: personal computers became ubiquitous in part because of the WIMP interface (not because it brought significantly more functionality to the household PC, but simply because it made it easy for the novice user to access functionality that previously required esoteric command-line instructions); file-transfer became easy with software such as Gnutella, Napster and Microsoft Messenger (though ftp and other low level mechanisms had existed for a long time); and browsing for information and resources became an everyday event with the easy availability of Netscape and Internet Explorer, even though systems such as Gopher (and even NCSA Mosaic) had been available and used by niche groups for some period. The current “early adopters” of the Grid — principally scientists who need to harness the power of distributed computing to solve current and pressing problems in their own fields — are technologically sophisticated and are motivated to make the required investment because they can see a potential payback now or in the near future. For more casual and collaborative use of the Grid to become commonplace — perhaps as an underlying infrastructure for generalised file sharing, multi-player gaming, application sharing (licensing issues aside) or as a collaborative distributed computing resource —

a more intuitive and friendly means of interaction is needed.

This paper discusses the issues involved in moving Grids from being a specialist tool for scientists to being generally available to the millions of computer users worldwide. Assuming a settling of the current technology into a robust, easily deployable toolkit, we concentrate instead on the problems of making the concepts of distributed computing meaningful to the non-specialist user, and of making the everyday management and use of Grid technology easy and intuitive to specialist and casual user alike. We briefly review current approaches to Grid user interfaces, and discuss their strengths and weaknesses. We then introduce the concept of the Grid-enabled desktop environment, and discuss its architectural implications and our current implementation.

2. Current Grid Interfaces

There are three main classes of Grid Computing Environments (GCE) [7] in current practice: command line interfaces, web portals and stand-alone clients.

Command-line interfaces, such as provided by the Globus Toolkit [8], are flexible and powerful, but are difficult to learn, and in particular are off-putting to a casual user familiar with today's point-and-click interfaces. Given that the motivation of Globus is to provide a standard toolkit on which others can build, it is quite reasonable that more thought has gone into the underlying Globus services and protocols than into user interfaces to them, and some inconsistencies in the various Globus commands persist.

Also worthy of mention is the proposed GCE shell [9]. Here the idea is to view the Grid as a distributed operating system, accessed through a consistent set of user-friendly high-level shell commands, which may in turn be programmed through scripts. In principle-given appropriate separation; such an approach can hide the details of whether the back-end infrastructure is based on Globus, Web services (e.g. [10]), "Grid services", and/or something else entirely (e.g. [11,12]).

Portals usually deliver their functionality either through a web browser (Web portals), or through bespoke, stand-alone "fat clients". They present friendlier interfaces to the user, while restricting access to particular sets of users and services. The kinds of services packaged in this way range widely, from generic job management portals such as CCLRC's HPCGrid Services Portal [13], to more application-specific portals (e.g. [14]). These share the typical, but not defining, characteristic of a user interface consisting of a series of forms delivered to the user's web browser, perhaps supported by Java applets or browser plug-ins. By routing traffic

through the portal, many problems with firewalls can be circumvented; it is easier to arrange to open ports on back-end systems for the portal (whose location is stable) than it is for a host of increasingly mobile client machines.

By exploiting the ubiquity of web browsers, the portal developer simplifies the problem of Graphical User Interface (GUI) distribution, and is able to expose powerful functionality to the user in a controlled way. However, by choosing what to expose, the developer is equally choosing what to hide—thus at the same time as being empowered, the user is also being constrained. The constraints are felt most severely when end users themselves (or third party application developers) want to develop their own interfaces to services on the Grid, perhaps with the aim of linking resources under the control of disjoint portals in complex workflows of their own devising. If it becomes common practice to hide Grid services behind portals, there is a real danger that power users will be reduced to "screen-scraping", a brittle and unreliable approach that has already caused chaos in the pre-Grid-services world of bio-informatics. On the positive side, there are encouraging signs that the portals community is recognising the importance of interoperability: enthusiastic acceptance of Web Services; emerging consensus on the idea of exposing middle tier services; and recent work on interoperable web portals.

The portal approach has many features to recommend it, but uniformity of presentation is not one of them. Although a user might be able to access many portals using the same Web browser, there is nothing to guarantee consistency of style, terminology and behaviour between any two portals.

Stand-alone "fat clients" run on the user's machine, and like web portals, range from domain-specific problem solving environments to fairly generic job and file management tools. The unicorn [15] client spans this range by providing a generic workflow management framework that allows the development of application-specific plug-ins.

Both stand-alone clients and Web portals (and, to a lesser extent, command-line interfaces) give rise to, and are liable to perpetuate, a perceptual gulf between "local" and "remote", the very thing that the Grid should be bridging. As all access to remote resources is "through" the portal, a clear distinction is drawn between the familiar local working environment and the distributed functionality of the Grid. This division complicates the user's conceptual model — local resources are "on here", the Grid is very much "out there"; and introduces practical hurdles in terms of interaction between local and remote tools. The sophistication of tools at both ends is often reduced to a lowest common factor by the

need for a significant amount of “cutting and pasting” when submitting jobs.

It does not require a great deal of sophistication to publish a web page, and millions of home users do so regularly. On the other hand, a home user offering Grid services to family and friends is unheard of today. As mentioned previously, we can imagine that Grid technology could be put to good effect in areas such as games [16], and file and application sharing (e.g. [17]); however global uptake of this technology will inevitably lead to more imaginative uses than we can predict here. By viewing the user as service consumer, portals tend to neglect the user as service provider; a trend that could turn the Grid into a more conventional client/server network than originally envisaged.

3. Grid-Enabling the Desktop

Designing an effective user interface is difficult, and requires considerable understanding of the tasks that the user may wish to perform, and of the user's mental model of the functionality of the technology (which may be very different from the details of the actual underlying implementation). An interface that invisibly maps “what I want to do” on to “what the technology actually does” is a pleasure to use; one that forces the user to think in unfamiliar terms, and to continuously translate their requirements onto the functions of the system is a nuisance, and is unlikely to gain widespread use beyond enthusiasts. The WIMP paradigm desktop interface is by far the most common example of a metaphor that “just works” for most purposes. Assuming the user understands that a computer is a machine capable of storing and of processing data, the metaphors such as “filing system”, “folder”, “document”, and “trashcan” lead the user quite intuitively through the functionality of device. “I can put my stuff in a document, file it in a folder, and dispose of it in the trash can” is a much more straightforward explanation than could ever be achieved by exposing details of the *inodes*, link structures and the working of the hard drive. There are many detailed design principles for making a user interface “easy to use”, such as penalty-free exploration of its functionality (e.g. using drop down menus allow a user to browse possible features without the cost of executing one that may do something unwanted) or the provision of consistent and complete *cancel* and *undo* functions, but these are secondary to determining a suitable metaphor to guide the overall structure of the interface. Applying this principle to Grid computing environments, we argue that though a Grid portal's interface may well exhibit exemplary *detailed* design, its metaphor of “accessing the Grid *through* a portal” is not

consistent with the metaphor of the Grid itself as a network of sharable and dynamically configurable resources, especially one in which the user's own machine may well participate.

As a move towards an interface metaphor that is more closely related to the idea of a Grid of distributed data and computing power, we present the notion of the Grid Enabled Desktop. The argument for suitability is this: to many users, their desktop environment *is* their computer. Most users will be happily unaware that there is a hard drive and a filing system acting to store their information (and their “tools”), and will only know of the CD-ROM as a distinct device because they feed media into it occasionally. Details such as the CPU and the RAM are only important to the more technically sophisticated user; to most they are simply a means of making the computer (*the desktop*) work. This close correspondence between the desktop and the computer means that to make the computer participate in a Grid, we should make the desktop participate. Notions such as *sharing* (whether as a consumer or provider) a particular folder, file, or application is likely to match more comfortably with a user's model of their machine than the issue of “setting up a server”, even though both may achieve the same end result.

To this end we are developing a GRid ENabled Desktop Environment (GRENADE), the architecture of which is described in the remainder of this paper.

4. The GRENADE Approach

By a Grid-enabled desktop, we do not simply mean a suite of GUI-based tools to augment the existing command line mechanisms; rather we are aiming at something more closely integrated with the user's everyday desktop environment, that enables interoperability with existing desktop applications, and picks up all the familiar desktop metaphors of *drag and drop*, *cut and paste* and so on. The process involves initially understanding how users “comprehend the Grid”, and then mapping artifacts from the users' conceptual model through onto underlying Grid-technology by means of appropriate metaphors on the desktop. Ideally, this modelling and mapping process should be driven by end-users and their tasks rather than by features of the Grid's infrastructure. However, our target audience of “casual Grid users” is currently limited, and we have had to base our initial prototype on our own predictions of how users may view the Grid, with the intention of refining the model through an iterative process of making prototypes freely available and then performing user studies.

We can identify a number of classes of task that the user may undertake: browse a directory of resources; browse a shared data repository;

define and submit a job; or access a service. Parallel to these activities will be the need to manage credentials. Our initial GRENADE prototype is therefore aimed at mapping these tasks onto suitable desktop metaphors.

Browsing a shared repository of data is perhaps the simplest of these activities to relate to the desktop environment, and (issues of availability and authorisation aside) should be no different to browsing through any other local or remote file store. It is common practice for “the browser” to be a window onto all manner of filing systems on most platforms, and plug-in modules to KDE's Konqueror and Microsoft's Explorer make these tools as happy displaying the contents of Web PagesTP sites, WebDAV repositories and so on, as they are showing local file systems; common operations such as search, arrange by date etc. are as applicable to remote data as they are to resources hosted on the user's own machine. Integrating GridFTP and MDS with these browser tools by means of the appropriate plug-in mechanism appears to be the ideal way of providing a familiar resource browsing mechanism and the current implementation is discussed later. Integration of GridFTP with the OS's filesystem is an alternative approach to providing this browsing capability, and this work is underway in a partner project, UTOPIA [18].

Job submission is currently a less familiar desktop task, though has parallels to the idea of sending a job to a printer: the parameters of the job are defined by some mechanism; the job is submitted, and the desktop environment provides feedback as to the progress of the job, as well as mechanisms for inspecting the print queue, modifying the behaviour of the print device etc. The important issue here is that the task of printing is embedded deeply in the desktop environment; one does not “open up a printing application” but accesses this service via a variety of routes embedded within other applications or by “dropping a document onto the printer”. The printer is an object in the user's model, and the desktop provides progress reports as required, and mechanisms that can be invoked for examining this object when necessary. It is this paradigm that we extend for submitting and monitoring Grid jobs with GRENADE.

The management of individual *credentials* is essentially a new addition to the desktop environment. Though users will almost certainly have multiple passwords to remember for logging on to machines, accessing web sites etc., and may use authentication systems such as Microsoft's Passport or its alternatives, the idea of managing multiple credentials as objective entities is likely to be alien.

Again, there is little by way of a defined application here; rather the functionality of storing credentials and prompting for passwords must be integrated with the general day-to-day

use of the desktop and its associated tools and applications, and rather than “single sign on” (which assumes a *one-credential-fits-all* scenario) we plan instead “authenticate on demand”, where the system intelligently associates credentials with services, and prompts the users for passwords and certificates when necessary.

5. Architectural Issues

The GRENADE infrastructure was designed to be as portable as possible, supporting KDE and Microsoft Windows initially, with support for other platforms possible in the future. A number of different architectures were considered, with three factors being particularly important: the mechanism for generating the Graphical User Interface in such a way that it integrates and blends in with the user's desktop; a means of accessing the Globus Toolkit — chosen because of its current popularity — which functions on both platforms; and a robust and secure mechanism for inter-process communication.

Choosing a widget set for the construction of the user interface was comparatively straightforward. Though Java provides cross-platform GUI features via its Swing classes, applications built with Swing have a distinct look-and-feel rather than blending in and inter-operating with other desktop applications (a noteworthy example of such an application is “The Grid Desktop Client” [19], though this is more of a Grid GUI than an attempt to closely integrate with the desktop). Many other GUI building toolkits such as GTK or Microsoft's Foundation Classes work reliably on only one or other of the platforms. Trolltech's C++ QT widget set, on the other hand, is implemented natively for both our target platforms, adopting the look and feel of its environment.

Providing a Windows implementation of GRENADE proved the greatest challenge since the only realistic version of Globus available on this platform at the time of writing is the Java COG Kit [20]. Integrating Java with native C or C++ is still a relatively complex and potentially error-prone affair; rather than proceed down this route (or adopting similar architectures utilising middleware of one form or another), we chose instead to rely on the `globusrun` command provided by both the Java COG and standard binary Globus distribution. Given that GRAM jobs generally have long execution times relative to the cost of launching an executable (even when taking into account the slow startup times of Java programs) the additional overhead introduced proves negligible.

Adopting this comparatively loose coupling between the GRENADE processes providing the user interfaces and the underlying Globus commands led us towards using straightforward text files as a form of interprocess

communication, resulting in an architecture for GRENADE reminiscent of traditional spooling configurations such as Unix print and email services. System-state becomes persistently stored in a particular file structure; user-agents integrated with the desktop environment provide a means for monitoring this state; and system-agents (*daemon* processes) invisibly manipulate this state behind the scenes via `globusrun` (provided either natively or via the COG Kit). This ostensibly crude form of IPC proved to have a number of important benefits over more complex message passing mechanisms:

- At all stages the messages are safely queued, with the robust nature of today's file systems providing data integrity. The system still works even when end users shutdown their machines or disconnect from the network. Even unexpected system failure is likely to cause the system to recover its files into a safe state. This is important as a Globus job could take days to complete and, for example, the user may take their laptop elsewhere in the meantime.
- Control of a particular file is passed between GRENADE processes using simple file-based locking and by exploiting the atomic nature of the "move/rename" filesystem function. This avoids the need for complex and non-portable IPC mechanisms, and suitable protocols for passing control from one process to another means that the system can fail-safe (e.g. an operation that was part way through executing when a catastrophic system failure causes it to abort simply resumes next time the process becomes active).
- Integrating other languages and applications with the GRENADE submission and monitoring GUI becomes trivial, since all that is necessary for a particular language or application is the ability to write a correctly formatted text file into the correct location (to prevent malformed files from being generated, it is our intention to provide the appropriate functions for a variety of popular programming and scripting languages, and though the format of the files will be made available, applications are to be discouraged from writing these files directly).

This approach also makes the operation of the underlying system more transparent. For example, operations, such as submitting a job, can fail for a variety of reasons and at different stages during the submission process. If the failed job's parameters have been recorded in an RSL file which persists on disk for the lifetime of the job rather than encoded in a now-unreachable API call, it is straightforward for a

developer or expert user to investigate and debug failed operations by manually trying to resubmit the job; a task that becomes much harder to debug without the actual RSL file being available for inspection. Secondly, a less complex "glue infrastructure" leads to a system that is both more robust and more flexible. For example, by supporting the native `globusrun` executable on Unix platforms the higher memory and startup costs associated with spawning Java programs can be avoided. Finally, the system is insulated from changing APIs within Globus since it is reasonable to expect the command line interface will remain the same or at least similar.

GRENADE relies on two types of desktop object, representing GRAM resource managers and GRAM jobs. Appearing on the screen as suitable icons, these are in fact simple text files and can be created through the desktop's standard template-based file creation mechanisms (fig.1). The user then edits these files using dedicated editor applications provided with GRENADE.

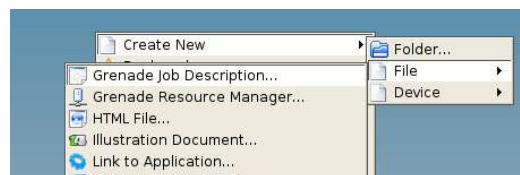


Fig.1 GRENADE object creation

Jobs are submitted by dragging and dropping the job description on the relevant resource manager (fig.2). On Windows this is accomplished using a Drop Handler shell extension (a COM object which is loaded into Explorer); on KDE it requires the reference to the resource manager to be stored in a ".desktop" application shortcut file.

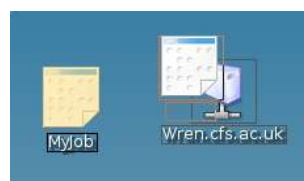


Fig.2 Job submission

Dropping the job on the resource manager causes a new file with a unique filename containing the contents of both files to be created and placed in GRENADE's job queue directory. A command line tool is also provided for queueing jobs from within scripts and other programs.

Having created this new request file, a second, empty lock file is created with a ".n" extension signifying that the new file is ready for submission and that ownership can be passed to the next program in the submission pipeline. Periodically a daemon polls the queue directory

for new “.n” files. When one is found, the daemon generates an RSL file that describes the job, including the necessary parameters to cause the `stdout` and `stdin` streams to be stored for later retrieval. The RSL file is then submitted by calling the `globusrun` command.

Whenever the `globusrun` executable is called, if the user does not have a proxy valid for at least an hour, a new one is generated (fig.3). On KDE, passwords for different credentials can be stored in the system “Wallet” [21], thereby providing a single sign-on facility across multiple Grids.



Fig.3 Pass phrase entry (if not already in wallet)

If the job is submitted successfully, the Job ID is stored in the original file and the lock file is renamed to have a “.p” extension signifying the job has been successfully submitted and is now pending.



Fig.4 Notification of submission

The system-tray monitor periodically polls pending jobs in order to check if they have completed. When they have, it fetches any `stdout` and `stderr` output and issues a notification message (fig.4) that the job has completed. By clicking on this icon the user is then able to see details of the job and view the output.

At all stages any error that occurs is written into the original file and the lock file is renamed to have a “.e” extension. This causes the error message to also be displayed by the system-tray monitor and from within the job manager.

Since all the system state is stored in persistent files on the disk, the submission computer can be shut down and rebooted without affecting the operation of the job. When the user logs back in any pending jobs will continue to be polled and the user will be notified of jobs that have completed in the meantime. Where users mount their home directories from a remote file store they are also able to continue to monitor pending jobs from a different machine.

The further aspect of Grid computing addressed by GRENADE is that of transparent access to remote data resources. For a number of

reasons the architecture used to implement this functionality is more tightly coupled to the desktop environment:

- The underlying mechanisms used to implement file access in different desktop environments varies considerably, e.g. KIO [22] for KDE, VFS [23] for Gnome. It would therefore be unlikely that the architecture specific code written for one might usefully be employed in providing the same functionality in another.
- The loose coupling to the Globus toolkit achieved by using the command line interface with its various implementations is not possible as this interface does not expose the full GridFTP functionality – e.g. it is not directly possible to perform a directory listing from the command line tools.

An obvious alternative to the command line interface offering a far richer API, whilst still retaining platform portability, is the Java CoG kit. A GRENADE GridFTP component based on the Java CoG kit [20] would in some ways be the ideal solution—the job management modules already use this as the Globus middleware layer on Windows and doing so on all platforms would obviate the need for a Globus installation even on Unix. Implementing a Java KIOSlave is, however, currently extremely non-trivial. A number of promising but ultimately unsuitable solutions were examined:

- JNI – the Java Native Interface (JNI [24]). A KIOSlave *stub* could create a Java Virtual Machine (JVM) and then forward method calls by JNI to a Java implementation using the CoG kit. This has the disadvantage that a significant amount of coding in C++ is still required and, more importantly, that the creation of a JVM varies considerably between different VMs and platforms.
- KDE Java bindings [25]. These allow Java code to make calls to KDE/QT libraries via native headers using the JNI. Whilst they are comprehensive, they are designed such that the flow of control must always originate from Java. This is not appropriate for KIOSlaves as the KIO sub-system must create the slave object, a process which here would involve the creation of a Java Virtual Machine. This functionality is currently not implemented.
- GCJ – A demonstration KIOSlave was created [26] by *compiling* Java code using the GNU Compiler for Java [27]. This allows binary distributions which link against `libgcj` – part of the GCC[28] suite and available on most Unix platforms. Unfortunately it was not possible to compile the CoG kit as this depends on certain security libraries with which GCJ is not yet compatible. In the near future, however, this

very useful technique should become possible.

Despite the lack of success in finding a Java based solution, a degree of abstraction has been achieved by providing a wrapper around the Globus C API to provide a simplified interface to GridFTP file management. The library is compiled as a shared object which can be utilised by code tied more closely to the desktop environment.

The KDE mechanism for abstracting file access, KIO, provides a pluggable interface whereby *KIOSlaves* implement support for different protocols. GRENADE provides such a *KIOSlave* for GridFTP. Remote resources can therefore be accessed simply by using a URL of the form:

```
gsiftp://<grid_ftp_host>/<path>/<filename>
```

Since all KDE applications operate with files using KIO, this allows files in a remote GridFTP repository to be manipulated in Konqueror (the file browser), loaded directly into applications, saved from them etc. In fact the GridFTP resource can be used exactly as a local hard drive (fig.5).



Fig.5 Loading a file with GridFTP KIOSlave - note the filename in the titlebar

6. Future Work

Work on GRENADE is continuing. The user interfaces shown presently provide a fairly low-level interface to defining Globus jobs. It is envisaged that eventually different Globus services will provide application specific job editors that will hide the complexity of command-line details from the end-users. A hypothetical example might be an oceanographic simulation that provides a graphical map to specify a region of interest and a number of sliders to set initial parameters for a simulation run. The result of the completed job would then be automatically displayed in a viewer application as a full 3D iso-surface, with the specifics of the simulation startup and file formats entirely hidden from the user.

The GridFTP component is currently the least mature part of GRENADE, providing a read-only interface. Whilst it is already a useful tool we plan to extend it to provide the complete KIO functionality. Of particular significance is the fact that the high level abstraction that KIO provides over file operations will permit more advanced features of GridFTP to be utilised. If a file is dragged from one remote GridFTP resource windows to another, the KIOSlave can

recognise this and automatically perform the operation as a third party transfer. It is this type of built-in, automatic *intelligence* that we believe to be the major benefit of the GUI, providing users with **sensible** choices by default without the requirement that they even understand the technologies involved.

The shared object wrapper library approach taken by the GridFTP KIOSlave marries well with the objectives of the Simple API for Grid Applications Research Group (SAGA-RG [29]) ratified just before GGF11 [30]. We believe that a positive engagement with this initiative could benefit both projects.

We plan to extend the functionality of our GRENADE system to include access to other existing Grid features such as resource discovery/browsing and file sharing. The close integration with the desktop demonstrated by the current job submission prototype not only provides an easy way of accessing otherwise complex technology, but opens up many possibilities by exploiting other desktop tools (for example, using an instant messaging client to inform a mobile user that a job submission has completed).

What is less clear at this early stage is how to map the more detailed functionality of the Grid onto the desktop metaphor. With present Grid technology, resources tend to be data/information, services (such as search engines or format conversion tools), or compute-engines capable of accepting arbitrary jobs of work. Though these loose categories may provide some insight into appropriate metaphors on the desktop (e.g. a data repository is a shared folder, a service is a shared application), it is likely that any distinction between these categories will blur as the Grid gains popularity (e.g. a service that produces dynamic data, a compute engine that tends to run specialist services etc.). It is also not completely clear at this stage exactly which aspects of the Grid toolkits to expose to the user: though concepts such as "job", "credential" and "service" may have common enough meaning to form part of a casual user's conceptual model, *gatekeepers* and *proxies* are likely to be of interest only to the more experienced user.

Grid security, and PKI in general, is a complex topic even for technical users of Globus, and is often cited as the most difficult challenge faced by new users. The current profusion of email-borne viruses and fake websites attempting to steal bank details demonstrate how even the most basic security can be a significant challenge to the average computer user. It is not yet clear how the complexities of the Grid's security infrastructure can be made comprehensible to our envisaged non-technical users, and issues such as how to tell whether a newly displayed password requester is *really* being displayed by

GRENADE, or some other rogue program, remain challenges to be addressed.

No amount of prediction on our part, however, will answer these questions conclusively, and thus GRENADE will be made freely available in order for the widest possible audience to participate in its future development.

7. Conclusion

The use of a Graphical User Interface in a Desktop Environment has been critical in facilitating the adoption of computers by non-technical users. GRENADE continues to explore how the concepts of the Grid can be expressed within this paradigm. It is developing a user interface foundation for grid applications complementary, but orthogonal, to both command line interfaces and web portals.

8. References

- [1] A.Szalay and J.Gray, *The World-Wide Telescope*, Science, 293,pp2037-2040,2001
- [2] K.Keahey, T.Fredian, Q.Peng, D.P.Schissel M.Thompson, I.Foster, M.Greenwald, D.McCune, *Computational Grids in Action: The National Fusion Collaboratory*, Future Generation Computer Systems, 18(8) pp1005-115, 2002
- [3] S.Brunett, K.Czajkowski, S.Fitzgerald, I.T. Foster, A.E.Johnson, C.Kesselman, J.Leigh, S.Tuecke, *Application Experiences with the Globus Toolkit*, HPDC, pp81-,1998
- [4] I.Foster, C.Kesselman, S.Tuecke, *The anatomy of The Grid: Enabling scalable Virtual Organisations*, Int. J. Supercomp. App., 15(3), 2001
- [5] *CORBA/IIOP specification*, 2001 <http://www.omg.org/cgi-bin/doc?formal/02-11-01.pdf>
- [6] *Open Software, Introduction to OSF DCE*, Prentice Hall, ISBN:0131858106, 1995
- [7] G.C.Fox, D.Gannon, M.Thomas, *A summary of grid computing environments*, Conc. & Comp.:Practice and Experience, 14(13), 2002
- [8] I.Foster, C.Kesselman, *Globus: A Metacomputing Infrastructure Toolkit*, Int. J. Supercomp. App. and High Perf. Comp., 11 (2) 115-128, 1997
- [9] *Developing a secure grid computing environment shell engine*, http://grid.ucs.indiana.edu/ptliupages/publications/npsec_gceshell.pdf, Community Grid Lab, Indiana University.
- [10] M.Pierce, C.Youn, G.Fox, S.Mock, K.Mueller, O.Balsoy, *Interoperable Web Services for Computational Portals*, Proc. ACM/IEEE Conf. on Supercomp., 2002
- [11] D.Abramson, R.Sosic, J.Giddy, B.Hall, *Nimrod: A Tool for Performing Parameterised Simulations Using Distributed Workstations*, HPDC, 112-121,1995
- [12] M.J.Litzkow, M.Livny, M.W.Mutka, *Condor - A hunter of idle workstations*, Proc. 8th Int. Conf. Dist. Comp. Sys, 104-111, 1988
- [13] *HPC Portal*, <http://tyne.dl.ac.uk/HPCPortal/>, 2004
- [14] M.Russel, G.Allen, G.Daues, I.Foster, E.Seidel, J.Novotny, J.Shalf, G.von Laszewski, *The Astrophysics Simulation Collaboratory: A Science Portal Enabling Community Software Development*, Cluster Computing, 5(3) 297-304, 2002
- [15] *UNICORE Forum*, <http://www.unicore.org/>, 2004
- [16] D.Levine, M.Wirt, B.Whitebook, *Practical Grid Computing for Massively Multiplayer Games*, ISBN 1584502924, 2004
- [17] M.Ripeanu, I.Foster, A.Iamnitchi, *Mapping the Gnutella Network: Properties of Large-Scale Peer-to-Peer Systems and Implications for System Design*, IEEE Internet Computing, 6(1), 2002
- [18] S.Pettifer, J.R.Sinnott, T.K.Attwood, *UTOPIA: user friendly tools for operating informatics applications*, Comparative and Func. Genomics, 5(1), 56-60, 2004
- [19] G.von Laszewski, *CoG Kit DnD Project Page*, <http://www-unix.globus.org/cog/projects/dnd>, Argonne National Laboratory
- [20] G.von Laszewski, *CoG kits: a bridge between commodity distributed computing and high-performance grids*, Proc. ACM Java Grande, 97-106, 2000
- [21] G.Staikos, *KWallet - The KDE Wallet System*, <http://www.staikos.net/~staikos/papers/2003/kwallet-kastle-2003.ps>, 2003
- [22] D.Faure, *KDE 2.0 as a development framework - KIO*, http://www.blackie.dk/~dfaure/OSDEM/html/slide_9.html
- [23] *Gnome VFS*, <http://www.gnu.org/directory/libs/GnomeVFS.html>
- [24] Koala, <http://developer.kde.org/language-bindings/java/qtjava.html>
- [25] JNI, <http://java.sun.com/j2se/1.3/docs/guide/jni/>
- [26] D.Hanlon, *A Java KIOSlave in KDevelop*, http://ralph.mvc.mcc.ac.uk/~djn/spip/article.php?id_article=11
- [27] *GCC*, <http://gcc.gnu.org/java/>
- [28] *GCC*, <http://gcc.gnu.org/>
- [29] *SAGA*, <https://forge.gridforum.org/projects/saga-rg/>
- [30] *GGF*, <http://www.gridforum.org/>